자료구조 Homework #6

충북대학교 컴퓨터공학과 2022040014 오재식

```c
//*
 *  doubly-linked-list.c
 *
 *  Doubly Linked List
 *
 *  Data Structures
 *  School of Computer Science
 *  at Chungbuk National University
 *
 */



#include<stdio.h>
#include<stdlib.h>

typedef struct Node {
    int key;
    struct Node* llink;
    struct Node* rlink;
} listNode;



typedef struct Head {
    struct Node* first;
}headNode;


int initialize(headNode** h);

int freeList(headNode* h);

int insertNode(headNode* h, int key);
int insertLast(headNode* h, int key);
int insertFirst(headNode* h, int key);
int deleteNode(headNode* h, int key);
int deleteLast(headNode* h);
int deleteFirst(headNode* h);
int invertList(headNode* h);

void printList(headNode* h);


int main()
{
    char command;
    int key;
    headNode* headnode=NULL;

    do{
```

```
        printf("------------------------------------------------------------------
\n");
        printf("                     Doubly Linked  List
\n");
        printf("------------------------------------------------------------------
\n");
        printf(" Initialize    = z           Print         = p \n");
        printf(" Insert Node   = i           Delete Node   = d \n");
        printf(" Insert Last   = n           Delete Last   = e\n");
        printf(" Insert First  = f           Delete First  = t\n");
        printf(" Invert List   = r           Quit          = q\n");
        printf("------------------------------------------------------------------
\n");

        printf("Command = ");
        scanf(" %c", &command);

        switch(command) {
        case 'z': case 'Z':
            initialize(&headnode);
            break;
        case 'p': case 'P':
            printList(headnode);
            break;
        case 'i': case 'I':
            printf("Your Key = ");
            scanf("%d", &key);
            insertNode(headnode, key);
            break;
        case 'd': case 'D':
            printf("Your Key = ");
            scanf("%d", &key);
            deleteNode(headnode, key);
            break;
        case 'n': case 'N':
            printf("Your Key = ");
            scanf("%d", &key);
            insertLast(headnode, key);
            break;
        case 'e': case 'E':
            deleteLast(headnode);
            break;
        case 'f': case 'F':
            printf("Your Key = ");
            scanf("%d", &key);
            insertFirst(headnode, key);
            break;
        case 't': case 'T':
            deleteFirst(headnode);
            break;
```

```
        case 'r': case 'R':
            invertList(headnode);
            break;
        case 'q': case 'Q':
            freeList(headnode);
            break;
        default:
            printf("\n       >>>>>   Concentration!!   <<<<<     \n");
            break;
        }

    }while(command != 'q' && command != 'Q');

    return 1;
}


int initialize(headNode** h) {

    // headNode가 null이 아니면 headeNode의 메모리를 해제 시킴
    if(*h != NULL)
        freeList(*h);

    // headNode에 메모리할당
    *h = (headNode*)malloc(sizeof(headNode));
    (*h)->first = NULL;
    return 1;
}

int freeList(headNode* h){
    /* h가 가리키고 있는 node를 p에 할당하고
    Prev에 p, p에 p->rlink 할당 후 prev 메모리해제하면서 리스트의 모든 메모리해제 후
    h 또한 메모리 해제해줌.
     */
    listNode* p = h->first;

    listNode* prev = NULL;
    while(p != NULL) {
        prev = p;
        p = p->rlink;
        free(prev);
    }
    free(h);
    return 0;
}


void printList(headNode* h) {
    int i = 0;
    listNode* p;
```

```
    printf("\n---PRINT\n");

    if(h == NULL) {
        printf("Nothing to print....\n");
        return;
    }

    p = h->first;

    while(p != NULL) {
        printf("[ [%d]=%d ] ", i, p->key);
        p = p->rlink;
        i++;
    }

    printf("  items = %d\n", i);
}




/**
 * list 의 마지막 위치에 key 삽입
 */
int insertLast(headNode* h, int key) {

    listNode* node = (listNode*)malloc(sizeof(listNode));
    node->key = key;
    node->rlink = NULL;
    node->llink = NULL;

    if (h->first == NULL) // 맨 앞이 null 이면 맨 앞에 삽입
    {
        h->first = node;
        return 0;
    }

    listNode* n = h->first;
    while(n->rlink != NULL) { // 마지막 노드 찾기
        n = n->rlink;
    }
    n->rlink = node; // 마지막 노드에 오른쪽에 삽입
    node->llink = n;
    return 0;
}



/**
```

```c
 * list 의 마지막 노드 삭제
 */
int deleteLast(headNode* h) {

    if (h->first == NULL)
    {
        printf("nothing to delete.\n");
        return 0;
    }

    listNode* n = h->first;
    listNode* trail = NULL;

    /* 노드의 오른쪽 노드가 존재하지 않는다면 first node == last node 이므로 n 을 메모리
해제*/
    if(n->rlink == NULL) {
        h->first = NULL;
        free(n);
        return 0;
    }

    /* 리스트의 마지막 노드 찾기 */
    while(n->rlink != NULL) {
        trail = n;
        n = n->rlink;
    }

    /* trail 의 오른쪽 노드가 N 이므로 n 에 값을 null 로 만들고 메모리해제 */
    trail->rlink = NULL;
    free(n);

    return 0;
}



/**
 * list 의 맨 앞에 key 를 삽입
 */
int insertFirst(headNode* h, int key) {

    listNode* node = (listNode*)malloc(sizeof(listNode));
    node->key = key;
    node->rlink = node->llink = NULL;

    if(h->first == NULL) // 처음노드가 없다면 처음 노드에 삽입
    {
        h->first = node;
        return 1;
```

```
    }

    node->rlink = h->first;
    node->llink = NULL;

    listNode *p = h->first;
    p->llink = node; // 기존 처음 노드 왼쪽에 노드 삽입
    h->first = node; // 삽입 노드를 h 가 가리키도록 함.

    return 0;
}

/**
 * list 의 처음 노드 삭제
 */
int deleteFirst(headNode* h) {

    if (h->first == NULL)
    {
        printf("nothing to delete.\n");
        return 0;
    }
    listNode* n = h->first; // 처음을 가리키는 포인터를 두번째 노드 값을 가리키도록 함.
    h->first = n->rlink;

    free(n); // 기존 노드 메모리 해제

    return 0;
}



/**
 * 由ь 뒳㉿몄쓽 留곹겕璵㉿ ㉿닚㉿쇰줉 ㉿ 諛곗튂
 */
int invertList(headNode* h) {


    if(h->first == NULL) {
        printf("nothing to invert...\n");
        return 0;
    }
    listNode *n = h->first;
    listNode *trail = NULL;
    listNode *middle = NULL;

    while(n != NULL){
        trail = middle;
        middle = n;
        n = n->rlink;
```

```
        middle->rlink = trail;
        middle->llink = n;
    }

    h->first = middle;

    return 0;
}



/*key 값이 기존 노드의 key 값과 비교 했을 때 기존 key 값이 더 크다면 그 앞에 노드 삽입.*/
int insertNode(headNode* h, int key) {

    listNode* node = (listNode*)malloc(sizeof(listNode));
    node->key = key;
    node->llink = node->rlink = NULL;

    if (h->first == NULL)
    {
        h->first = node;
        return 0;
    }

    listNode* n = h->first;

    /* key 값이 기존 노드의 key 값보다 작은지를 확인 */
    while(n != NULL) {
        if(n->key >= key) {
            /* n 이 first 노드 값과 같다면 inserFirst 함수 사용 */
            if(n == h->first) {
                insertFirst(h, key);
            } else { /* n 과 n->llink 사이에 node 삽입 */
                node->rlink = n;
                node->llink = n->llink;
                n->llink->rlink = node;
            }
            return 0;
        }

        n = n->rlink;
    }

    /* 반복문이 끝난 후에도 함수가 종료되지 않았다면 inserLast 를 사용하여 삽입 */
    insertLast(h, key);
    return 0;
}


/**
```

```c
 * list 에서 key 값이 존재하면 삭제
 */
int deleteNode(headNode* h, int key) {

    if (h->first == NULL)
    {
        printf("nothing to delete.\n");
        return 1;
    }

    listNode* n = h->first;

    while(n != NULL) {
        if(n->key == key) {
            if(n == h->first) { /* 삭제하려는 노드가 처음 값이라면 함수사용 */
                deleteFirst(h);
            } else if (n->rlink == NULL){ /* 마지막 노드라면 함수사용 */
                deleteLast(h);
            } else { /* n 을 리스트에서 제거 */
                n->llink->rlink = n->rlink;
                n->rlink->llink = n->llink;
                free(n);
            }
            return 1;
        }

        n = n->rlink;
    }

    printf("cannot find the node for key = %d\n", key);
    return 1;
}
```

```
---PRINT
[ [0]=4 ] [ [1]=2 ] [ [2]=1 ]   items = 3
-------------------------------------------------------------
                    Doubly Linked  List
-------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
-------------------------------------------------------------
Command = i
Your Key = 3
-------------------------------------------------------------
                    Doubly Linked  List
-------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
-------------------------------------------------------------
Command = p

---PRINT
[ [0]=3 ] [ [1]=4 ] [ [2]=2 ] [ [3]=1 ]   items = 4
-------------------------------------------------------------
                    Doubly Linked  List
-------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
-------------------------------------------------------------
Command = q
➜  자료구조 git:(main) ✗
```

```
To https://github.com/ohjaesik/homework7.git
   0f25cb2..d153d7d  main -> main
→  자료구조  git:(main) ✗ git add doubly-linked-list.c
→  자료구조  git:(main) ✗ git commit -m '이름 출력'
[main 29244c7] 이름 출력
 Committer: 오재식 <ojaesig@ojaesigs-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 2 insertions(+), 1 deletion(-)
→  자료구조  git:(main) ✗ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 362 bytes | 362.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ohjaesik/homework7.git
   d153d7d..29244c7  main -> main
→  자료구조  git:(main) ✗
```

https://github.com/ohjaesik/homework7